Logical Operations and Truth Tables

At first glance, it may not seem that the study of logic should be part of mathematics. For most of us, the word logic is associated with reasoning in a very nebulous way:
"If my car is out of gas, then I cannot drive it to work."
seems logical enough, while

"If I am curious, then I am yellow."

is clearly illogical.

Yet our conclusions about what is or is not logical are most often unstructured and subjective. The purpose of logic is to enable the logician to construct valid arguments which satisfy the basic principle

"If all of the premises are true, then the conclusion must be true."

It turns out that in order to reliably and objectively construct valid arguments, the logical operations which one uses must be clearly defined and must obey a set of consistent properties. Thus logic is quite rightly treated as a mathematical subject.

Up until now, you've probably considered mathematics as a set of rules for using numbers. The study of logic as a branch of mathematics will require you to think more abstractly then you are perhaps used to doing. For instance, in logic we use variables to represent **propositions** (or premises), in the same fashion that we use variables to represent numbers in algebra. But while an algebraic variable can have any number as its value, a logical variable can only have the value **True** or **False**. That is, True and False are the "numerical constants" of logic. And instead of the usual arithmetic operators (addition, subtraction, etc.), the logical operators are "AND", "OR", "NOT", "XOR" ("eXclusive OR"), "IMPLIES" and "EQUIVALENCE". Finally, rather than constructing a series of algebraic steps in order to solve a problem, we will learn how to determine whether a statement is always true (a **tautology**) or is a **contradiction** (never true), and whether an argument is valid.

Truth Tables

In algebra, it is rarely possible to guess the numerical solution to a problem, and because there are an infinite number of numbers it is obvious that one cannot try all possible solutions in order to find one that solves the problem. But in logic, we only have two "numbers": True and False. Therefore, any logical statement which contains a finite number of logical variables (which of course covers any problem we have to deal with) can be analyzed using a table which lists all possible values of the variables: a "**truth table**". Since each variable can take only two values, a statement with "n" variables requires a table with 2^n rows. Using the letters "p", "q", "r", etc., to represent logical variables, we can construct truth tables for statements involving any number of variables (although we will usually limit ourselves to at most three variables per statement to simplify the matter):

p
T
F

for statements with one variable,

p	**q**
T	T
T	F
F	T
F	F

for statements with two variables and

p	**q**	**r**
T	T	T
T	T	F
T	F	T
T	F	F
F	T	T
F	T	F
F	F	T
F	F	F

for statements with three variables (where in every case, "T" stands for True and "F" for False). The extension to more than three variables should now be obvious:

1. for the first variable, the first half of the rows are T while the second half are F
2. for the second variable, the rows are split into four sections: the first and third quarters are T while the second and fourth quarters are F
3. for the third variable, the rows are split into eighths, with alternating eighths having T's and F's
4. in general, for the nth variable, the rows are split into 2^n parts, with alternating T's and F's in each part

Logical Operators

We will now define the logical operators which we mentioned earlier, using truth tables. But let us proceed with caution: most of the operators have names which we may be accustomed to using in ways that are fuzzy or even contradictory to their proper definitions. In all cases, use the truth table for an operator as its exact and only definition; try not to bring to logic the baggage of your colloquial use of the English language.

The first logical operator which we will discuss is the "**AND**", or **conjunction** operator. For the computer scientist, it is perhaps the most useful logical operator we will discuss. It is a "**binary**" operator (a binary operator is defined as an operator that takes two operands; not binary in the sense of the binary number system):

p AND q

It is traditionally represented using the following symbol:

$\wedge$

but we will represent it using the ampersand ("**&**") since that is the symbol most commonly used on computers to represent a logical AND. It has the following truth table:

p	**q**	**p & q**
T	T	T
T	F	F
F	T	F
F	F	F

Notice that p & q is only T if both p and q are T. Thus the rigorous definition of AND is consistent with its colloquial definition.

The **OR** (or **disjunction**) operator is also a binary operator, and is traditionally represented using the following symbol:

$\vee$

We will represent OR using the stroke ("∨"), again due to common usage on computers. It has the following truth table:

p	**q**	**p ∨ q**
T	T	T
T	F	T
F	T	T
F	F	F

p ∨ q is true whenever either p is true, q is true or both p and q are true (so it too agrees with its colloquial counterpart).

The **NOT** (or **negation** or **inversion**) operator is a "**unary**" operator: it takes just one operand, like the unary minus in arithmetic (for instance, -x). NOT is traditionally represented using the tilde ("~")

In a programming environment, NOT is frequently represented using the exclamation point ("**!**"). Since the exclamation point is too easy to mistake for the stroke, we will use the tilde instead. Not has the following truth table:

p	**~ p**
T	F
F	T

~ p is the negation of p, so it again agrees with its colloquial counterpart; it is essentially the 1's complement operation.

The **XOR** (eXclusive OR) operator is a binary operator, and is not independent of the operators we have presented thus far (many texts do not introduce it as a separate logical operator). It has no traditional notation, and is not often used in programming (where our usual logical operator symbols originate), so we will simply adopt the "**X**" as the symbol for the XOR:

p	**q**	**p X q**
T	T	F
T	F	T
F	T	T
F	F	F

p X q is T if either p is T or q is T, **but not both**. We will see later how to write it in terms of ANDs, ORs and NOTs.

The implication operator (**IMPLIES**) is a binary operator, and is defined in a somewhat counterintuitive manner (until you appreciate it, that is!).

but we will denote it with an arrow ("**->**"):

p	**q**	**p -> q**
T	T	T
T	F	F
F	T	T
F	F	T

So p -> q follows the following reasoning:

1. a True premise implies a True conclusion, therefore T -> T is T;
2. a True premise cannot imply a False conclusion, therefore T -> F is F; and
3. you can conclude anything from a false assumption, so F -> anything is T.

IMPLIES (**implication**) is definitely one to watch; while its definition makes sense (after a bit of thought), it is probably not what you are used to thinking of as implication.

EQUIVALENCE is our final logical operator; it is a binary operator, and is traditionally notated by either an equal sign, a three-lined equal sign or a double arrow ("**<->**"):

p	**q**	**p <-> q**
T	T	T
T	F	F
F	T	F
F	F	T

p <-> q is T if p and q are the same (are equal), so it too follows our colloquial notion.

Just as with arithmetic operators, the logical operators follow "**operator precedence**" (an implicit ordering). In an arithmetic expression with sums and products and no parentheses, the multiplications are performed before the additions. In a similar fashion, if parentheses are not used, the operator precedence for logical operators is:

1. First do the NOTs;
2. then do the ANDs;
3. then the ORs and XORs, and finally
4. do the IMPLIES and EQUIVALENCEs.

Needless to say, it will behoove you to use parentheses whenever you have any doubts!